31장. RegExp

31.1 정규 표현식이란?

1. **정규 표현식(regular expression)**

   : 일정한 패턴을 가진 문자열의 집합을 표현하기 위해 사용하는 형식 언어다.

2. 정규 표현식은 문자열을 대상으로 **패턴 매칭 기능**을 제공.

   패턴 매칭 기능이란, 특정 패턴과 일치하는 문자열 검색 or 추출 or 치환할 수 있는 기능

[예제 31-01]

```
// 사용자로부터 입력받은 휴대폰 전화번호
const tel = '010-1234-567팔'

// 정규 표현식 리터럴로 휴대폰 전화번호 패턴을 정의한다.
const regExp = /^\d{3}-\d{4}-\d{4}$/;

// tel이 휴대폰 전화번호 패턴에 매칭하는지 테스트(확인)한다.
regExp.test(tel);  // false
```

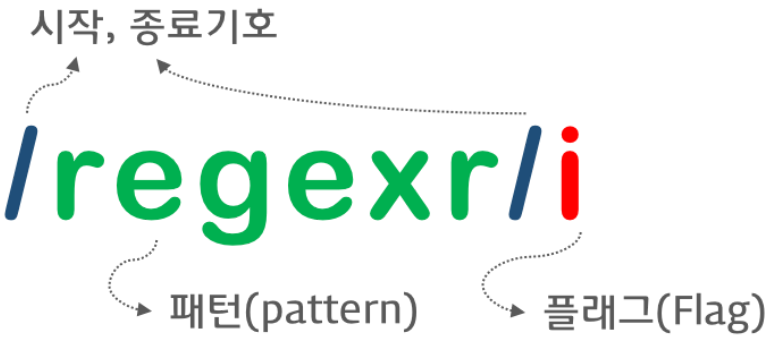
3. 정규 표현식의 장단점

장점 : 반복문, 조건문 없이 패턴 정의하고 테스트하는 것으로 간단히 체크할 수 있다.

단점 : 주석, 공백 허용치 않고 여러 가지 기호를 혼합하여 사용하기 때문에 가독성 좋지 않다.

31.2 정규 표현식의 생성

1. 그림 3-11 정규 표현식 리터럴



- 정규 표현식 리터럴은, 패턴과 플래그로 구성된다.

[예제 31-02]

```
const target = 'Is this all there is?';

// 패턴 : is
// 플래그 : i  => 대소문자 구별하지 않고 검색
const regexp = /is/i;

// test 메서드는 target 문자열에 대해 정규 표현식 regexp의 패턴을 검색하여 매칭
// boolean 값으로 반환
regexp.test(target);  // true
```

2. [예제 31-03]

```
const target = 'Is this all there is?';

const regexp = new RegExp(/is/i); // ES6

regexp.test(target);  // true
```

- RegExp 생성자 함수 사용

31.3 RegExp 메서드

31.3.1 RegExp.prototype.exex

1. [예제 31-05]

```
const target = 'Is this all there is?';
const regexp = /is/;

regexp.exec(target);
// [ "is", index : 5, input : "Is this all there is?", groups : unde
```

- 문자열 내의 모든 패턴을 검색하는 g플래그를 지정해도 첫 번째 매칭 결과만 반환

31.3.2 RegExp.prototype.

1. [예제 31-06]

```
const target = 'Is this all there is?';
const regexp = /is/;

regexp.test(target); // true
```

- 인수로 전달받은 문자열에 대해 정규 표현식의 패턴을 검색하여 매칭 결과를 반환

31.3.3 String.prototype.match

1. [예제 31-07]

```
const target = 'Is this all there is?';
const regexp = /is/;

target.match(regExp);
// [ "is", index : 5, input : "Is this all there is?", groups : unde
```

- 대상 문자열과 인수로 전달받은 정규 표현식과의 매칭 결과를 배열로 반환

2. exec 메서드와의 차이점

- exec 메서드 : 문자열 내의 모든 패턴을 검색하는 g플래그를 지정해도 **첫 번째 매칭 결과**만 반환
- match 메서드 : g플래그가 지정되면 **모든 매칭 결과**를 배열로 반환.

31.4 플래그

1. 플래그는 총 6개.

플래그	의미	설명
i	Ignore case	대소문자를 구별하지 않고 패턴 검색
g	Global	대상 문자열 내에서 패턴과 일치하는 모든 문자열을 전역 검색
m	Multi line	문자열 행이 바뀌더라도 패턴 검색을 계속한다.
s		`.`이 개행 문자 `\n`도 포함하도록 'dotall' 모드를 활성화한다.
u		유니코드 전체를 지원한다. 이 플래그를 사용하면 서로게이트 쌍(surrogate pair)을 올바르게 처리할 수 있다.
y		문자 내 특정 위치에서 검색을 진행하는 'sticky' 모드를 활성화 시킨다.

- 플래그는 옵션이므로 선택적 사용 가능
- 순서 상관없이 하나 이상의 플래그를 동시에 설정할 수 있다.
- 플래그 사용하지 않는 경우, 대소문자를 구별해서 패턴을 검색.
- 매칭 대상이 1개 이상 존재해도, 첫 번째 매칭 대상만 검색하고 종료.

31.5 패턴

31.5.1 문자열 검색

1. 대수문자 구별하지 않고 검색하려면 플래그 i 사용

2. 검색 대상 문자열 내, 패턴과 일치하는 모든 문자열을 전역 검색하려면 플래그 g 사용

31.5.2 임의의 문자열 검색

1. `.`은 임의의 문자 한 개를 의미.
    - 문자의 내용은 상관 X

[예제 31-13]

```
const target = "Is this all there is?";

// 임의의 3자리 문자열을 대소문자 구별하여 전역 검색
const regExp = /.../g;

target.match(regExp); // [ "Is ", "thi", "s a", "ll ", "the", "re ",
```

31.5.3 반복 검색

1. {m,n}은 앞선 배턴이 최소 m번, 최대 n번 반복되는 문자열을 의미.

```
const target = 'A AA B BB Aa Bb AAA';

// A가 최소 1번, 최대 2번 반복되는 문자열을 전역 검색한다.
const regExp = /A{1,2}/g;

target.match(regExp); ['A', 'AA', 'A', 'AA', 'A']
```

```
const target = 'A AA B BB Aa Bb AAA';

// A가 2번 반복되는 문자열을 전역 검색한다.
const regExp = /A{2}/g;

target.match(regExp); ['AA', 'AA']
```

```
const target = 'A AA B BB Aa Bb AAA';

// A가 최소 2번 이상 반복되는 문자열을 전역 검색한다.
const regExp = /A{2,}/g;

target.match(regExp); ['AA', 'AAA']
```

2. +는 앞선 패턴이 최소 한번 이상 반복되는 문자열을 의미

   + === {1, }

3. ?는 앞선 패턴이 최대 한 번(0번 포함) 이상 반복되는 문자열을 의미

   ? === {0, 1}

31.5.4 OR 검색

1. | 은 or의 의미를 갖는다.
   - 분해되지 않는 단어 레벨로 검색하기 위해서는 +를 함께 사용

- 범위를 지정하려면 [ ] 내에 -를 사용

2. \d는 숫자를 의미, \d === [0-9]

   \D는 \d와 반대로 동작, 즉 숫자가 아닌 문자 의미

3. \w는 알파벳, 숫자, 언더스코어를 의미, \w === [A-Za-z0-9_]

   \W는 반대로 동작, 즉 알파벳, 숫자, 언더스코어가 아닌 문자 의미.

31.5.5 NOT 검색

1. [...] 내의 ^은 not의 의미를 갖는다.

   [^0-9]는 숫자를 제외한 문자를 의미.

31.5.6 시작 위치로 검색

1. [...] 밖의 ^은 문자열의 시작을 의미.

31.5.7 마지막 위치로 검색

1. $는 문자열의 마지막을 의미.

31.6 자주 사용하는 정규표현식

31.6.1 특정단어로 시작하는지 검사

1. ‘http://’ 또는 ‘https://’로 시작하는지 검사

```
const url = 'https://example.com';

/^https?:\/\//.test(url); // true
;
/^(http|https):\/\//.test(url); // true
```

31.6.2 특정단어로 끝나는지 검사

1. 대상 문자열이 ‘html’로 끝나는지 검사

```
const fileNmae = 'index.html';

/html$/.test(fileName); // true
```

31.6.3 숫자로만 이루어진 문자열인지 검사

```
const target = "12345";

/^\d+$/.test(target); // true
```

31.6.4 하나 이상의 공백으로 시작하는지 검사

1. 하나 이상의 공백으로 시작하는지 검사

```
const target = ' Hi';

/^[\s]+/.test(target); // true
```

31.6.5 아이디로 사용 가능한지 검사

1. 알파벳 대소문자 or 숫자로 시작하고 끝나며 4~10자리인지 검사.

```
const id = 'abc123';

/^[A-Za-z0-9]{4,10}$/.test(id); // true
```

31.6.6 메일 주소 형식에 맞는지 검사

1. 메일 주소 형식에 맞는지 검사

```
const email = 'ungmo2@gmail.com';

/^[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*\
```

31.6.7 핸드폰 번호 형식에 맞는지 검사

```
const cellphone = '010-1234-5678';

/^\d{3}-\d{3,4}-\d{4}$/.test(cellphone); // true
```

31.6.8 특수 문자 포함 여부 검사

1. 특수 문자가 포함돼 있는지 검사.

```
const target = 'abc#123';

// A-Za-z0-9 이외의 문자가 있는지 검사한다.
(/[^A-Za-z0-9]/gi).test(target); // true
```

2. 특수문자 제거 → String.prototype.replace 메서드 사용

```
target.replace(/[^A-Za-z0-9]/gi, ''); // abc123
```